Machine Learning

Luke Min

February 9, 2024

Abstract

Based on notes from Spring 2024 CIS 5200 at Penn.

1 Week 1

1.1 January 18, 2024

Introduction and Overview

- Everything is on Canvas
- CIS 4190/5190: Applied Machine Learning
- CIS 5450 Big Data Analytics - Data Science. Build a machine learning system by yourself, e.g. for a startup
- ESE 5450 Data Mining - Data Science + Math.

You can take CIS 5200 and CIS 5450.

Prerequisites, Homework 0, etc.

Grades

- 6HWs (55%), Midterm (20%), Final (25%)
- Can submit multiple tries for Gradescope. Instant feedback for coding assignments.

What is machine learning?

- Improve **performance**
- At some **task**
- Given **experience**

Types of Machine Learning

- Supervised learning
- Unsupervised learning
- Semi-supervised
- Active / Online / Reinforcement Learning, etc....

2 Week 2

2.1 January 23, 2024

Typical ML Pipeline: Data -> Learning Algorithm -> Knowledge

- Data: Experience, Training Data
- Learning: Optimization
- Knowledge: Performance on some task, a model

Choices to make

1. Which Data?
2. Which Algorithm?
3. Pick Hyperparameters, estimate
4. Evaluate or test the model.

Learning Paradigms

1. **Supervised Learning**:
   (a) Training Data $= (x_i, y_i) = (\text{Data}, \text{Labels})$
   (b) Goal is to train a model that gets labels
2. **Unsupervised Learning**:
   (a) No labels. No strict performance metric.
3. **Semi-Supervised Learning**
   (a) Mix of labelled and unlabelled data.
4. **Online Learning**:
   (a) Learning by sequentially taking one data point at a time.
5. **Active Learning**:
   (a) You get to choose the next data point you observe.
6. **Reinforcement Learning**:
   (a) Collect their own data, make their own decisions

Supervised Learning. Our notation for the dataset will be $\mathcal{D}$ which we write as

$$\mathcal{D} = \{(x_1, y_1), \cdots, (x_i, y_i), \cdots, (x_n, y_n)\}$$

Typically the $x_i \in \mathcal{X}$ will be **inputs**, $y_i \in \mathcal{Y}$ the **labels**, and n the number of data points. We call $\mathcal{X}$ the **feature space** (typically $\mathbb{R}^d$) and the $\mathcal{Y}$ the **label space**.

We will also frequently work with the **feature matrix** X and the **label vector** Y which we write as

$$X = \begin{bmatrix} -x_1- \\ -x_2- \\ \vdots \\ -x_n- \end{bmatrix} \in \mathbb{R}^{n \times d} \quad \text{and} \quad Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \in \mathcal{Y}^n$$

Classification Example: Cat vs. Dog. In this case, the dataset $\mathcal{D}$ consists of images of cats or dogs (x_i's) and the labels (y_i). Here the x_i = RGB Pixels. If the images are 1000x1000 pixels and RGB has three values, then $d = 1000 \times 1000 \times 3$. The labels could be written as

$$y_i = \begin{cases} +1 & \text{if dog} \\ -1 & \text{if cat} \end{cases}$$

This is called a (binary) classification problem. In a multi-class classification problem we can say

$$y_i = \begin{cases} (1,0,0) & \text{if dog} \\ (0,1,0) & \text{if cat} \\ (0,0,1) & \text{if otter} \end{cases}$$

Regression Example: Housing Prices. If you want to build a model that predicts housing prices, the dataset $\mathcal{D}$ consists of descriptors of sold houses, and the price the house sold at. x_i could be number of beds or baths, age, size, and y_i could be the price. This is called a regression problem.

Measuring Performance with Error. A model $h : \mathcal{X} \to \mathcal{Y}$ is a function from the feature space to the label space, which outputs a predicted value $\hat{y} = h(x)$. Thus, for each data x_i, we want to compare the predicted value $\hat{y}_i = h(x_i)$ to the actual value y_i. We write the metric used as $\hat{R}$ for **risk** which is

$$\hat{R}_{0/1}(h) = \frac{1}{n}\sum_{i=1}^{n} l_{0/1}(h(x_i), y_i)$$

where the 0/1 loss function

$$l_{0/1}(\hat{y}_i, y_i) = \begin{cases} 0 & \text{if } \hat{y} = y \\ 1 & \text{otherwise} \end{cases}$$

This is usually referred to as **training error of h.** In the case of regression, we can replace the zero one loss with squared loss,

$$l_{sq}(\hat{y}_i, y_i) = (\hat{y}_i - y_i)^2$$

All of machine learning is basically trying to estimate $h(\cdot)$ by minimizing risk.

Generalization Gap. Once you estimate $h(\cdot)$ on training data, we use a separate set of test data to test the the model $h(\cdot)$. The issue is that our model may not generalize to unseen test data. We try to quantify this as the generalization gap which we write as |Training Error − Test Error|.

2.2 January 25, 2024

How do we choose h? If we were to minimize training risk, we could always memorize the entire training data set by

$$h(x) = \begin{cases} y_i & \text{if } x = x_i \\ 1 & \text{otherwise} \end{cases}$$

This extreme example leads to the issue of *generalization.*

Generalization. Suppose we have $(x_i, y_i) \sim \mathcal{P}$ the training data. If in addition we have $(x, y) \sim \mathcal{P}$ then we can define the **true risk** as

$$R(h) = \mathbb{E}_{(x,y)\sim\mathcal{P}}\left[l(h(x), y)\right]$$

We want to minimize true risk, not training risk. Now we can write

$$R(h) = \left[\underbrace{R(h) - \hat{R}(h)}_{\text{gap}}\right] + \hat{R}(h)$$

Because we want to minimize true risk $R(h)$, but only observe training risk $\hat{R}(h)$, one thing we could do is set things up so that generalization bound holds, i.e. $R(h) - \hat{R}(h) \leq f(n)$ where $f(n) \to 0$ as $n \to \infty$. With infinite data, we can basically close the gap. But we return to this later.

Theoretically, the best we can do is minimize the true risk:

$$\arg\min_h R(h) = h^*$$

We call this h the **Bayes Optimal Classifier**.

Example: Binary Classification. Let's go through a simple example of binary classification. If we know

$$\eta(x) = P(Y = 1|\, x)$$

and use the zero one loss $l_{0/1}$ then we get

$$\begin{aligned} R(h) &= \mathbb{E}_{(x,y)\sim\mathcal{P}}\left[l_{0/1}(h(x), y)\right] \\ &= \mathbb{E}_x\left[\mathbb{E}_{y|x}\left[l_{0/1}(h(x), y)\right]\right] \\ &= \mathbb{E}_x\left[P(Y = 1|\, x)\,\mathbb{I}(h(x) = -1) + P(Y = -1|\, x)\,\mathbb{I}(h(x) = 1)\right] \\ &= \mathbb{E}_x\left[\eta(x)\,\mathbb{I}(h(x) = -1) + (1 - \eta(x))\,\mathbb{I}(h(x) = 1)\right] \end{aligned}$$

In the third line, the $0/1$ loss becomes 1 only when h gets the choice wrong.

The choice of $h = h^*$ that minimizes true risk is

$$h^*(x) = \begin{cases} -1 & \text{if } \eta(x) < 0.5 \\ +1 & \text{otherwise} \end{cases}$$

Hence the true risk in this case becomes

$$R(h^*) = \mathbb{E}_x\left[\min(\eta(x), 1 - \eta(x))\right]$$

where we assumed the particular distribution $\eta(x)$ and the loss function $l_{0/1}$.

We call the true risk under Bayes Optimal Classifier the **Bayes Error.**

k-Nearest Neighbors. Suppose we have X's and O's distributed over some space. k-nearest neighbors is an estimator we can use when we *assume that data points close to each other of the same class.*

We take some subset of the training data $N_x \subseteq \mathcal{D}$, which we call the neighborhood of x. Formally we assume that $\forall z \in \mathcal{D}\backslash N_x$, $z' \in N_x$ we have

$$d(x, z) \geq d(x, z')$$

In other words, given some neighborhood of x that we call N_x, all the points in that neighborhood is close to x than any point outside of it.

The **k-Nearest Classifier** is

$$h(x) = \text{mode}\{\, y' |\, (x', y') \in N_x\}$$

One key way this might fail is if the differently labelled data points are clustered together.

Another potential issue is that in high dimensional space, the distances are very far apart. Roughly it might seem like all the points are equally apart, and the KNN CLF spits out something that looks like it was randomly chosen.

Another issue is if our distance function does not capture our distance appropriately. For example, if we take $d(x, z) = \|x - z\|_2^2$ or the Euclidean distance, it might not work very well for some form of data.

3 Week 3

3.1 January 30, 2024 - Perceptron

Last lecture, we talked about k-nearest neighbors. The algorithm builds neighborhoods N_x that make the same predictions for labels. Issues may arise in: high dimensions (all points seem equidistant) and selecting an appropriate notion of a distance (e.g. what is a good metric for images?).

Today we discuss **perceptrons**, which we build on to get to deep learning.

Building on k-nearest neighbors, we could instead draw a line that separates the space into differently labelled regions. We call this a **linear predictor**. Perceptron is an algorithm that helps us find this linear predictor.

Lines. Suppose we consider a hyperplane in $\mathbb{R}^d$ defined by

$$w^\top x + b = 0$$

where $x, w \in \mathbb{R}^d$. Note $w^\top x = \|w\| \|x\| \cos\theta$.

- If $w^\top x > 0$ then x is on the same side as the region that w is pointing to
- If $w^\top x < 0$, then x is on the other side.

Hence w basically defines a boundary for x's.

Linear Model (w, b). The linear model for classification is given by where

$$h(x) = \operatorname{sgn}\left(w^\top x + b\right).$$

The function

$$\operatorname{sgn}(a) = \begin{cases} +1 & \text{if } a \geq 0 \\ -1 & \text{if } a < 0 \end{cases}$$

gives the "sign" of its argument.

The linear model retains some helpful properties:

Scale Invariance. If we replace the linear model (w, b) with $(\alpha w, \alpha b)$ for some $\alpha > 0$, nothing changes, as all we care about is the angle. So we can set $\|w\| = 1$.

Suppose we have a dataset $(x_i, y_i)_{i=1}^m$. Let $\alpha = \max_i \|x_i\|$ and normalize

$$\tilde{x}_i = \frac{x_i}{\alpha}$$

which gives us the condition that $\|\tilde{x}_i\| \leq 1$.

In practice, we start by scaling the dataset first to satisfy $\|\tilde{x}_i\| \leq 1$, and then find w in that space.

Bias b is useless. Suppose we define new variables as

$$\tilde{x} = \begin{bmatrix} x \\ 1 \end{bmatrix} \quad \text{and} \quad \tilde{w} = \begin{bmatrix} w \\ b \end{bmatrix}$$

which gives us $\tilde{w}^\top \tilde{x} = w^\top x + b = 0$. The tilde transformation adds an extra dimension, and slides the whole plane down by b. Now in this new space, the $\tilde{x}$ goes through the origin. This tells us that we can always redefine the problem so that the relevant subspace goes through the origin.

Margin γ. In words, the margin γ is defined as the minimum distance of x_i to the line defined by w:

$$\gamma := \min_i \left| w^\top x_i \right|$$

This means that given a linear model (w, b) if we have margin γ, there are no points within γ distance of the linear model.

The margin shows how much room you have on either side, and tells you how much space for errors you have. If the classifier has a good margin, then its a good classifier. If the margin is very small, then its a challenging problem.

$y_i w^\top x_i > 0$**.** If the sign of y_i and $w^\top x_i$ are the same, then we have $y_i w^\top x_i > 0$. In other words, we have $y_i w^\top x_i > 0$ when the prediction is correct. Hence we can use the quantity $y_i w^\top x_i$ to proxy for correctness. If the quantity is very large and positive, then we are very far from the margin, and we are quite sure about that point; if it's very negative, then we place a lot of confidence that it's correct, but it is in fact incorrect.

Perceptron. We first state the assumption behind the perceptron algorithm.

The realizability / Linear Separability assumption: $\exists w_*$ such that $\forall x_i$, $y_i w_*^\top x_i > 0$. $\gamma_{w^*} = \gamma$. This assumes that there is some "perfect" classifier.

Here's the algorithm:

1. Start with $w_0 = 0$
2. For $t = 1, 2, \cdots$ find i such that $y_i w_t^\top x_i \leq 0$. Update $w_{t+1} = w_t + y_i x_i$.
3. Repeat. If None, break.

Why does this algorithm work? Assume $\|x_i\| = 1$. If we get that $y_i w_t^\top x_i \leq 0$, that means we got that point wrong. And as we update,

$$w_{t+1} = w_t + y_i x_i$$

so that

$$\begin{aligned} w_{t+1}^\top x_i &= \left(w_t + y_i x_i\right)^\top x_i \\ &= w_t^\top x_i + y_i \left\|x_i\right\|^2 \\ &= \begin{cases} w_t^\top x_i + \left\|x_i\right\|^2 & \text{if } y_i > 0 \\ w_t^\top x_i - \left\|x_i\right\|^2 & \text{if } y_i < 0 \end{cases} \end{aligned}$$

Closely observe the above under two cases. First, when $y_i = +1$, then, $w_t^\top x_i \leq 0$, and updating the w this way "adds" to w_t so that it is now closer to being positive (which was the correct prediction). Similarly, the opposite holds for $y_i = -1$. Since we got it wrong ("$y_i w_t^\top x_i \leq 0$"), this means $w_t^\top x_i \geq 0$, and we need to bring it down somehow. That's what happens in the updating step.

Adding a wrong point and updating w rotates the whole w to the "correct" place.

Convergence Proof. A measure of closeness to w_* is given by $w_*^\top w_t$. So that can increase if either (1) the angle is getting smaller; or (2) norm of w_t is getting larger. So a better measure is

$$\frac{w_*^\top w_t}{\|w_t\|} = \cos\theta_t$$

Now given an update at t, substitute

$$\begin{aligned} w_*^\top w_t &= w_*^\top (w_{t-1} + y_i x_i) \\ &= w_*^\top w_{t-1} + y_i \left(w_*^\top x_i\right) \end{aligned}$$

Note $y_i \left(w_*^\top x_i\right) > 0$ since w_* is a perfect classifier. We also know $\left|w_*^\top x_i\right| \geq \gamma$. So we have

$$\begin{aligned} w_*^\top w_t &\geq w_*^\top w_{t-1} + \gamma \\ &\geq w_*^\top w_{t-2} + 2\gamma \\ &\vdots \\ &\geq \underbrace{w_*^\top w_0}_{w_0 := 0} + t\gamma = t\gamma \end{aligned}$$

Thus we conclude

$$w_*^\top w_t \geq t\gamma$$

Now we need to show the denominator is not increasing too fast. The denominator is

$$\begin{aligned} \|w_t\|^2 &= w_t^\top w_t \\ &= (w_{t-1} + y_i x_i)^\top (w_{t-1} + y_i x_i) \\ &= \underbrace{w_{t-1}^\top w_{t-1}}_{=\|w_{t-1}\|^2} + 2\underbrace{y_i \left(w_{t-1}^\top x_i\right)}_{<0} + \underbrace{y_i^2 \left(x_i^\top x_i\right)}_{\leq 1} \\ &\leq \|w_{t-1}\|^2 + 1 \end{aligned}$$

Note $y_i \left(w_{t-1}^\top x_i\right) < 0$ because we always pick the "incorrectly predicted points" i to update the w. We have $y_i^2 \left(x_i^\top x_i\right) \leq 1$ because $y_i^2 = 1$ and the norm of x_i is bounded above by 1 by construction.

Continuing we get that

$$\begin{aligned} \|w_t\|^2 &\leq \|w_{t-1}\|^2 + 1 \\ &\leq \|w_{t-2}\|^2 + 2 \\ &\vdots \\ &\leq \|w_0\|^2 + t \\ &= t \end{aligned}$$

Now collecting our two conditions, we have

$$w_*^\top w_t \geq t\gamma \quad \text{and} \quad \|w_t\|^2 \leq t$$

Going back to our angle definition,

$$\begin{aligned} \cos\theta_t &= \frac{w_*^\top w_t}{\|w_t\|} = \frac{\geq \gamma t}{\leq t} \\ &\geq \frac{\gamma t}{\sqrt{t}} = \gamma\sqrt{t} \end{aligned}$$

Now since cosine is bounded, we get that

$$1 \geq \gamma\sqrt{t}$$

so that

$$t \leq \frac{1}{\gamma^2}$$

Caveats. When we define an algorithm, it's good to understand when it fails.

- When we have the realizability assumption fail, then the algorithm fails.
- When we have differently labelled data points in each region ("XOR Problem") it fails the assumption, and the algo doesn't work.

3.2 February 1, 2024 - Gradient Descent

Recap of Perceptron. Recall the perceptron algorithm:

1. Set $w_0 = 0$.
2. For $t = 1, 2, \cdots$
   (a) if $\exists i$ such that $y_i w_t^\top x_i \leq 0$ then $w_{t+1} = w_t + y_i x_i$
   (b) else break

Key takeaways of the perceptron algorithm:

1. We require that $\exists w_*$ such that $\forall i$, $y_i w_*^\top x_i \geq 0$.
2. Define the margin as
$$\gamma = \min_i \left| w_*^\top x_i \right|$$
where we assume that $\|w_*\| = 1$. The guarantee is that if you run the perceptron algorithm, it will terminate after at most $1/\gamma^2$ steps.

Note that it's possible to have $\gamma = 0$, and the guarantee could in theory be a terrible one. This algorithm, just like KNN, relies on the "geometry" of the problem. It's very specific to the structure of the problem, whereas other algorithms are more general purpose, like gradient descent.

Overview. Today we discuss the **gradient descent**, one of the most successful algorithms in machine learning. A lot of modern machine learning applications utilize a version of gradient descent. It works surprisingly well even in contexts where you wouldn't expect it to perform that well. We will see that even the perceptron algorithm that we saw last lecture is the same as gradient descent under appropriately defined objective functions.

What is our goal? Recall that we have the empirical risk
$$\min_h \hat{R}(h) = \frac{1}{n} \sum_{i=1}^{n} l\left(h\left(x_i\right), y_i\right)$$
where h is the model and l is the loss function. We considered two different loss functions. First the zero-one loss for classification:
$$l_{0/1}\left(\hat{y}, y\right) = \begin{cases} 0 & y = \hat{y} \\ 1 & \text{otherwise} \end{cases}$$
And least squares loss for regression
$$l_{sq}\left(\hat{y}, y\right) = \frac{1}{2}\left(\hat{y} - y\right)^2$$

Function Class H What are the function classes we consider? Note for the linear models we had
$$H = \left\{x \mapsto w^\top x + b\right\}$$
so mapping this to the minimization problem we get
$$\min_{w,b} \frac{1}{n} \sum_{i=1}^{n} l\left(w^\top x_i + b, y_i\right)$$
This process of restricting our function class H is called **parametrization**. Now the problem becomes more concrete! For a lot of loss functions, we can optimize this!

General Setup A more general setup would be

$$\begin{aligned} &\min_{w} F(w) \\ \text{such that} \quad & w \in C \end{aligned}$$

where F is the objective; C is the constraint set; and w is our choice variable. How can we solve this?

Let's pretend for now that F is simple. For instance, for small v we can approximate linearly as in a Taylor approximation:

$$F(w+v) \approx F(w) + \underbrace{\nabla F(w)^{\top}}_{\text{gradient}} v$$

Here we are assuming that $F(\cdot)$ is *locally linear*. This is the general idea of gradient descent.

Suppose $F(w+v) \leq F(w)$. Then

$$\begin{aligned} F(w+v) &= F(w) + \nabla F(w)^{\top} v \\ &= F(w) + \nabla F(w)^{\top} (-\eta \nabla F(w)) \\ &= F(w) - \eta \|\nabla F(w)\|^2 \end{aligned}$$

where we used $v = -\eta \nabla F(w)$, which guarantees $-\eta \|\nabla F(w)\|^2 \leq 0$.

Gradient Descent. Here we present the graident descent algorithm:

1. Set $w_0 \in \mathbb{R}^d$.
2. For $t = 1, 2, \cdots, T$
   (a) $w_{t+1} = w_t - \eta \nabla F(w_t)$
   (b) $F(w_{t+1}) = F(w_t) - \eta \|\nabla F(w_t)\|^2$
3. Stop when $\|\nabla F(w)\| \leq \epsilon$

Note that

$$\begin{aligned} F(w_{t+1}) &= F(w_t - \eta \nabla F(w_t)) \\ &= F(w_t) + \nabla F(w_t)^{\top} (-\eta \nabla F(w_t)) \\ &= F(w_t) - \eta \|\nabla F(w_t)\|^2 \end{aligned}$$

How should we choose the **learning rate** η? If it is too large, it will "jump over" the minimum point and miss it, and result in **divergence**. If it is too small, the learning will take too long. There are a number of strategies people have taken.

- Take a big first step, and then take many small steps.
- Start with a big η, and then decrease it over time by setting $\eta_{t+1} = 0.99\eta_t$.
- Transformers increase η in the beginning, keep it at that max for a while, and then linearly decrease it.
- etc..

Limitations of Gradient Descent. In what settings will gradient descent struggle to perform?

- Whenever the function $F(\cdot)$ is "flat" for some regions, the algorithm will get struck.
- When we have local minima, it will also get stuck there.

In general, when we stop making "local progress" the algorithm will terminate, i.e. $\nabla F(w_t) \approx 0$. This can happen at (i) local minimum, (ii) local maxima, (iii) saddle point. But what we want is the global minima.

For this class, we will assume there is a global minimum. In case of quadratic functions, this will be true.

In cases of non-differentiable functions, we can use sub-gradients. But for this class, we won't use it. For instance

$$\text{ReLu}(x) = \max\left(w^\top x, 0\right)$$

is used a lot it neural networks.

Perceptron and Gradient Descent Consider the loss function

$$l(\hat{y}, y) = \max(-\hat{y}y, 0)$$

If you run this gradient descent with $\eta = 1$, then you get the perceptron algorithm.

What's happening here? $\hat{y}y > 0$ when we get the correct prediction, in which case the max function is binding and loss is zero. On the other hand, $\hat{y}y < 0$ when we predict wrongly, and the loss is equal to $-\hat{y}y$.

Second Order Approximation. Consider the first and second order approximations:

1. $F(w+v) \approx F(w) + \nabla F(w)^\top v$
2. $F(w+v) \approx F(w) + \nabla F(w)^\top v + \frac{1}{2} v^\top H v$ where $H_{ij} = \frac{\partial^2 F}{\partial w_i \partial w_j}$ is the Hessian

How should we proceed in the second case? Take the derivative with respect to v to obtain

$$\nabla F(w) + Hv = 0$$

or that

$$v = -H^{-1} \nabla F(w)$$

An algorithm that uses this is called the **Newton's Method**. Now we no longer have the learning rate η.

Newton's Method. Here's the actual algorithm.

1. Set $w_1 \in \mathbb{R}^d$
2. For $t = 1, \cdots, T$
   (a) $w_{t+1} = w_t - H^{-1} \nabla F(w_t)$

This works really well if the function is actually close to quadratic. If it's exactly quadratic, then this gets you the solution in a single step.

The cost is that we now have to compute the Hessian, and then invert it. In high dimensions, this becomes quite costly. This is why in many cases we just use the gradient descent in practice.

When to Use Gradient Descent. When is it a good idea to use gradient descent?

- When the function is convex
- When the function is smooth i.e. if v is small then $|f(w+v) - f(w)|$ is small.

Convexity. When we restrict ourselves to a certain class of functions, then we can make some guarantees about the performan ce of gradient descent. **Convex functions** is an important one, and there are a number of definitions. Here's one.

A function F is **convex** if for all $\alpha \in [0,1]$ and for all $w, w' \in \mathbb{R}^d$,

$$F\left(\alpha w + (1-\alpha) w'\right) \leq \alpha F(w) + (1-\alpha) F(w')$$

If F is differentiable, then F is convex if for all $w, w' \in \mathbb{R}^d$,

$$F(w') \geq F(w) + \nabla F(w)^\top (w' - w)$$

If F is twice differentiable, then we require

$$\nabla^2 F(w) \succeq 0$$

or that the Hessian is the positive semi-definite. The Hessian is the rate of change of the gradient.

4 Week 4

4.1 February 6, 2024 - Gradient Descent, Empirical Risk Minimization

Quick Recap. In gradient descent, the goal is to learn a model $h(\cdot)$ as follows:

1. Pick some parameterized h. For example, take $h(x) = w^\top x + b$ linear.
2. Start with some $w_1 \in \mathbb{R}^d$.
3. For $t = 1, 2, \cdots$, compute $w_{t+1} = w_t - \eta_t \nabla F(w_t)$ for some loss $F(\cdot)$.

This works well under certain assumptions like convexity, which has several definitions:

1. (General) $F\left(\alpha w' + (1-\alpha) w\right) \leq \alpha F(w') + (1-\alpha) F(w)$ for all $w, w' \in \mathbb{R}^d$ and $\alpha \in [0,1]$.
2. (F is differentiable) $F(w') \geq F(w) + \nabla F(w)^\top (w' - w)$
3. (F is twice differentiable) $\nabla^2 F(w) \succeq 0$ (i.e. PSD), or for all non-zero $v \in \mathbb{R}^d$, $v^\top \nabla^2 F(w) v \geq 0$, or Hessian has non-negative eigenvalues.

A second property to keep in mind is L-smoothness. A function is smooth if its gradient does not change too fast.

1. $\|\nabla F(w) - \nabla F(w')\| \leq L \|w - w'\|$
2. $F(w') \leq F(w) + \nabla F(w)^\top (w' - w) + \frac{L}{2} \|w' - w\|^2$

The first condition essentially claims that if the inputs w and w' are quite close, then the gradient also has to be sufficiently close. When L is large, then the function is *less smooth.* If $L = 0$ for instance, then the gradient stays the same. We pick the smallest L such that this condition is true for all $w, w' \in \mathbb{R}^d$. If a function is 2-smooth, then it's also 10-smooth.

The second condition basically says the function should lie below a quadratic term. The function is upper bounded by some quadratic term. Constant L modulates how quadratic that second term is.

If we combine convexity and smoothness, it provides both some upper bound and some lower bound, "sandwiching" the function.

Strong convexity is like the opposite of smoothness. It says that the function's gradient should change by at least some rate, and says the function should have some curvature (i.e. it can't just be flat).

Now we present a theorem on the convergence of the gradient descent algorithm:

Theorem 1. *For a L-smooth convex function F, gradient descent with $\eta_t = 1/L$, we have*

$$F(w_{t+1}) - F(w_*) \leq \frac{L}{2t} \|w_1 - w_*\|^2$$

where w_* is the optimal value.

This theorem says that our distant to the optimal value at iteration $t+1$, given by $F(w_{t+1}) - F(w_*)$, can be bounded above. Let say we want the bound to be ϵ so that $\frac{L}{2t}\|w_1 - w_*\|^2 \leq \epsilon$. Then note that we need

$$t = \frac{L}{2\epsilon}\|w_1 - w_*\|^2$$

The algorithm itself does not depend on w_*, but this theorem gives us some form of theoretical guarantee for convergence.

Proof. First, we begin by showing $F(w_{t+1}) \leq F(w_t)$. Use the update condition given by

$$w_{t+1} = w_t - \frac{1}{L}\nabla F(w_t)$$

for $\eta = 1/L$. Smoothness gives us

$$\begin{aligned} F(w_{t+1}) &\leq F(w_t) + \nabla F(w_t)^\top (w_{t+1} - w_t) + \frac{L}{2}\|w_{t+1} - w_t\|^2 \\ &= F(w_t) - L(w_{t+1} - w_t)^\top (w_{t+1} - w_t) + \frac{L}{2}\|w_{t+1} - w_t\|^2 \\ &= F(w_t) - \frac{L}{2}\|w_{t+1} - w_t\|^2 \end{aligned}$$

We want to get rid of the gradient term, so we substitute the rearrange update condition $L(w_{t+1} - w_t) = -\nabla F(w_t)$ in the second line. This tells us that we will be decreasing the function at each iteration.

Second, we want to show how close we are getting to the minimizer w_*, i.e. w_t is getting closer to w_*. We use the definition of convexity on w_* and w_t:

$$\begin{aligned} F(w_*) &\geq F(w_t) + \nabla F(w_t)^\top (w_* - w_t) \\ &= F(w_t) - L(w_{t+1} - w_t)^\top (w_* - w_t) \\ &= F(w_t) + \frac{L}{2}\left[\|w_{t+1} - w_*\|^2 - \|w_t - w_{t+1}\|^2 - \|w_t - w_*\|^2\right] \end{aligned}$$

where we complete the square[1] using $2a^\top b = \|a+b\|^2 - \|a\|^2 - \|b\|^2$.

Now we combine the two results:

$$\begin{aligned} F(w_{t+1}) &\leq F(w_t) - \frac{L}{2}\|w_{t+1} - w_t\|^2 \\ -F(w_*) &\leq -F(w_t) - \frac{L}{2}\left[\|w_{t+1} - w_*\|^2 - \|w_t - w_{t+1}\|^2 - \|w_t - w_*\|^2\right] \end{aligned}$$

where we multiplied the second condition by (-1). We now add the two inequalities to obtain

$$\begin{aligned} F(w_{t+1}) - F(w_*) &\leq -\frac{L}{2}\|w_{t+1} - w_t\|^2 - \frac{L}{2}\|w_{t+1} - w_*\|^2 + \frac{L}{2}\|w_t - w_{t+1}\|^2 + \frac{L}{2}\|w_t - w_*\|^2 \\ &= -\frac{L}{2}\|w_{t+1} - w_*\|^2 + \frac{L}{2}\|w_t - w_*\|^2 \\ &= +\frac{L}{2}\left(\|w_t - w_*\|^2 - \|w_{t+1} - w_*\|^2\right) \end{aligned}$$

[1]Let $a = w_{t+1} - w_t$ and $b = -(w_* - w_t)$ so that

$$\begin{aligned} 2a^\top b &= \|a+b\|^2 - \|a\|^2 - \|b\|^2 \\ &= \|(w_{t+1} - w_t) - (w_* - w_t)\|^2 - \|(w_{t+1} - w_t)\|^2 - \|-(w_* - w_t)\|^2 \\ &= \|w_{t+1} - w_*\|^2 - \|w_{t+1} - w_t\|^2 - \|w_t - w_*\|^2 \end{aligned}$$

From here, we proceed by computing telescoping sums. To do that, we sum over $t = 1, \cdots, T$ as follows

$$\begin{aligned}\sum_{t=1}^{T} F(w_{t+1}) - TF(w_*) &\leq \frac{L}{2}\left(\|w_1 - w_*\|^2 - \|w_{t+1} - w_*\|^2\right) \\ &\leq \frac{L}{2}\|w_1 - w_*\|^2\end{aligned}$$

since $\|w_{t+1} - w_*\|^2 \geq 0$. Rearranging,

$$\sum_{t=1}^{T} F(w_{t+1}) - TF(w_*) \leq \frac{L}{2}\|w_1 - w_*\|^2$$

Recall that $F(w_{t+1}) \leq F(w_t)$ for each t, so that as t increases each iteration, F is at least non-increasing. Then, fixing T, we have that for all $t \leq T$, we have $F(w_{T+1}) \leq F(w_t)$. Hence, we can give an even lower bound

$$TF(w_{T+1}) - TF(w_*) \leq \sum_{t=1}^{T} F(w_{t+1}) - TF(w_*) \leq \frac{L}{2}\|w_1 - w_*\|^2$$

Thus in the end we get

$$F(w_{T+1}) - F(w_*) \leq \frac{L}{2T}\|w_1 - w_*\|^2$$

□

In the case of μ-strong convexity and L-smoothness, we can get even better results (check notes).

Smoothness makes sure you're not overshooting. Convexity ensures that you get to the solution.

4.2 February 8, 2024 - Linear Regression, Logistic Regression

Recall that the gradient descent applies to general algorithms. In particular, we haven't really specified what the objective function is, or what are hypothesis class is. Today, we first link it to the perceptron algorithm, and then generalize to different models.

Quick Recap: Perceptron. Recall our previous setting for the perceptron algorithm: inputs $x \in \mathcal{X} = \mathbb{R}^d$, labels $y \in \mathcal{Y} = \{-1, +1\}$, hypothesis class $x \mapsto \text{sign}\left(w^\top x + b\right)$, loss: $l(\hat{y}, y) = 0$ if $\hat{y} = y$, else 1. ("0/1 loss").

Logistic Regression. Instead of mapping to ± 1, we could map to the continuous interval $[0, 1]$ as a way to estimate probabilities: $x \mapsto \Pr(y = 1|\, x) \in [0, 1]$. Previously, we used the $\text{sgn}(\cdot)$ function. Now we can use the **squashing function**

$$\sigma(\cdot) : \mathbb{R} \to [0, 1]$$

resulting in the new hypothesis class

$$x \mapsto \sigma\left(w^\top x + b\right) \in [0, 1]$$

To be more specific, we will be using the **sigmoid function** that is defined as

$$\sigma(v) := \frac{1}{1 + e^{-v}}$$

note that

- $\lim_{v \to -\infty} \sigma(v) = 0$
- $\sigma(0) = 1/2$
- $\lim_{v \to \infty} \sigma(v) = 1$

We could multiply this by a constant to steepen or flatten the function. The steeper it is, harder it is for the gradient descent algorithm to converge.

One reason to do this (rather than the perceptron) would be to make our function continuous, so that we could apply the gradient descent method. Another reason is to handle the noise in the data. Recall that the perceptron algorithm assumes that the data is perfectly separable, which may not hold in many cases. This way, we can interpret the output as a probability, and handle cases where things are not perfectly separable.

Logistic Loss Function. Note that we can no longer use the 0/1 loss, since that will output a loss of 1, unless your predicted value is exactly equal to the label (i.e. $\hat{y} = y$). Our output of the sigmoid function will never be exactly 0 or 1 – this will only happen in the limit. So we need another function.

We will use something called the **logistic loss** as

$$l(\hat{y}, y) = \begin{cases} -\log(\hat{y}) & \text{if } y = 1 \\ -\log(1 - \hat{y}) & \text{if } y = -1 \end{cases}$$

The logistic loss is an upper bound on the 0/1 loss when graphed on the $l(\hat{y}, y)$ and $y \cdot w^\top x$ space.

Let's try to understand what this function does. Our predicted values are restricted to the interval $\hat{y} \in [0, 1]$. First, suppose $y = 1$. In the correct case where $\hat{y}$ is almost 1, then this loss will be zero; if it is instead we are very wrong and $\hat{y}$ is almost 0, then this loss will be tending to $+\infty$. Second, let's say the correct label is $y = 0$. Then if $\hat{y}$ is also close to zero, then $-\log(1 - \hat{y}) \approx 0$ as well. If $\hat{y} \to 1$, then $-\log(1 - \hat{y}) \to +\infty$.

In general, we would like this loss function to (i) be larger when we are wrong; and (ii) be smaller when we are closer to being correct.

An equivalent way to state this would be

$$l(h(x), y) = \log\left(1 + \exp\left(-y \cdot w^\top x\right)\right)$$

The whole goal of this is to frame this so that you are maximizing the probability of your data.

Probabilistic Perspective. In general, we can take two routes in machine learning: (1) Pick a loss function, and just minimize that loss; or (2) Formulate a probability distribution, and then maximize the probability of seeing my data. Sometimes the two are equivalent, which is going to be the case today. But not always.

Checking the Logistic Loss. Let us consider each case, and confirm that the loss function we wrote down is actually equiavalent.

First, let's say $y = 1$. Then,

$$\begin{aligned} -\log(\hat{y}) &= -\log\left(\left(1 + e^{-w^\top x}\right)^{-1}\right) \\ &= \log\left(1 + e^{-w^\top x}\right) \\ &= l(h(x), 1) \end{aligned}$$

Second, if $y = -1$, then

$$\begin{aligned}
-\log\left(1-\hat{y}\right) &= -\log\left(1-\frac{1}{1+e^{-w^\top x}}\right) \\
&= -\log\left(\frac{e^{-w^\top x}}{1+e^{-w^\top x}}\right) \\
&= \log\left(\frac{1+e^{-w^\top x}}{e^{-w^\top x}}\right) \\
&= \log\left(1+e^{w^\top x}\right) \\
&= l\left(h\left(x\right), -1\right)
\end{aligned}$$

So this confirms our previous claim.

Maximum Likelihood. Let us denote $\mathcal{L}$ the likelihood function, S the sample of the dataset, and θ the parameters. Then,

$$\begin{aligned}
\mathcal{L}\left(\theta\right) &= P\left(S\middle|\,\theta\right) \\
&= \prod_{i=1}^{m} P\left(x_i, y_i\middle|\,\theta\right) && \text{iid assumption} \\
&= \prod_{i=1}^{m} P\left(y_i\middle|\,x_i, \theta\right) \cdot P\left(x_i\middle|\,\theta\right)
\end{aligned}$$

Given some sample S, we want to maximize our probability of observing it. Hence the **maximum likelihood estimator** θ is given by the θ that solves the following:

$$\max_{\theta} \mathcal{L}\left(\theta\right)$$

The argmax of the above is equivalent to the argmax of the log-likelihood, which is

$$\max_{\theta} \log \mathcal{L}\left(\theta\right) = \max_{\theta} \sum_{i=1}^{m} \left[\log P\left(y_i\middle|\,x_i, \theta\right) + P\left(x_i\middle|\,\theta\right)\right]$$

MLE in Logistic Regression. In the logistic regression case we have that $P\left(y_i\middle|\,x_i, \theta\right) = \sigma\left(w^\top x + b\right)$ and $\theta = (w, b)$. Assume that $P\left(x_i\middle|\,\theta\right)$ is constant. Then,

$$\begin{aligned}
\max_{\theta} \log \mathcal{L}\left(\theta\right) &= \log \prod_{i=1}^{m} \frac{1}{1+\exp\left(-y \cdot w^\top x\right)} \\
&= -\sum_{i=1}^{m} \log\left(1+\exp\left(-y \cdot w^\top x\right)\right)
\end{aligned}$$

5 Week 5

5.1 February 13, 2024 - Linear Regression, Logistic Regression

5.2 February 15, 2024 - Linear Support Vector Machines

6 Week 6

6.1 February 20, 2024 - Nonlinear Modeling, Kernels

6.2 February 22, 2024 - Kernels, Kernel SVMs

7 Week 7

7.1 February 27 - Midterm

7.2 February 29 - TBD

8 Week 8

8.1 March 5 - Spring Break

8.2 March 7 - Spring Break

9 Week 9

9.1 March 12 - Neural Networks I

9.2 March 14 - Neural Networks II